ハイパフォーマンス
コンピューティング 79－4
（1999. 12. 3）

並列処理言語CLIPの実装と評価

柳瀬龍郎　奥川峻史(福井大学工学部)

〒910-8507 福井市文京三丁目9-1

E-mail: {yanase, okugawa}@radio.fuis.fukui-u.ac.jp

並列処理言語において情報を交換する方法には大きく分けて、配列などを共有メモリに配して行う方法と、MPIなどを使ったメッセージパッシングによる二通りの方法がある。また実際の言語においてはCやFORTRANなどでも、関数の引数や返値としても情報のやりとりが可能である。本研究では、これら従来の情報交換の方法よりも使いやすく、プログラムの可読性の向上を目的とした、futureによる関数レベルでの並列処理の実現をめざした並列処理言語CLIPの処理系の実装を行った。実装は並列計算機システムCLIPer(18CPU)、SUN WS(4CPU)およびDEC WS(6CPU)に対して行った。さらにいくつかのサンプルプログラムによるこれらのシステムでの実行性能評価を行ったところ、自動並列コンパイルでは見られなかった速度向上率が得られた。

Implementation and Evaluation of Parallel Language CLIP

Tatsuro Yanase, Syunji Okugawa

Depertment of Information Science

Faculty of Engineering, Fukui university

In order to develop parallel programs, we have two methods in ordinary apploach for inter-proseccer communication, using an array on shared memory and MPI for pessage passing. And, C and FORTRAN can exchnge an information as arguments in subprogram invocation and as a return value. We implemented parallel language CLIP based on C that is more suitable to make easily and to increase readability of a parallel/concurrent program. The CLIP allows using the future function that proposed in multilisp. We report preliminary evaluation for the inpremantation CLIP to CLIPer with 18 CPU's, workstations with 4 and 5 CPU's.

1. まえがき

マルチプロセッサシステムにおける並列処理のための、言語としてのアプローチの方法には、コンパイルするときのループの検出などによる自動並列化が、商用化されたシステムでのメジャーな手段となっている。また、これとは異なり、反復処理は少ないが複雑であるような、プログラムに対する並列化のための手段としては、実験レベルでは幾つかの方法が提案されているが、その一つに、関数（あるいはサブルーチン）単位での並列実行が考えられる。元来、関数やサブルーチンは機能的に纏まった処理を行うために用いられるが、これを処理の纏まりとして、並列処理の単位とすることは、アルゴリズムをコーディングする際にも、極めて有効になると考えられる。著者のひとり柳瀬が提案したCLIP[2]はmultilisp[1]のfutureをCに持ち込んだ関数単位の並列処理を実現する言語である。

本報告では著者らによって構築したローカルメモリを持つ共有メモリマルチプロセッサCLIPerと商用SMP計算機において、CLIPの言語処理系を設計、実装し,幾つかの例題プログラムを用いて評価を行ったのでその結果を報告する。

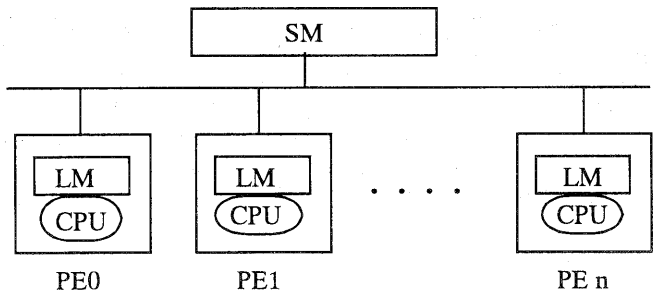

図１　CLIPer

2. CLIP

先にも触れたが、近年のSMPシステムのコンパイラは自動並列化オプションを備えているが、並列処理効率を良くするために、システムに依存する可能性の強い、細かいチューニングを自動並列化オプションに施すより、ユーザレベルの並列化機能を実現することも考慮すべきである。特に、ループ処理などにその威力を見せる自動並列化オプション等による方法では、効率的に並列処理が出来ないようなアルゴリズムを実現したアプリケーションにおいては、陽に並列化を指示する手段をプログラマに与えることが必要となる。このようなプログラミング環境においては、コンパイラによる支援が期待できないために、できるだけプログラマの負担を軽くすることも考慮しなければならない。

本研究ではこれらの背景をもとに、手続型言語Cにこの関数単位の並列処理メカニズムの導入を試みた。一般的に考えれば、Cにこの概念を持ち込む場合、また特にSMP計算機の場合にはスレッドを使えば容易にその実装が可能であるようにも思える。しかし、高並列あるいは分散システムでのスケーラビリティを想定した場合、システム内の全てのメモリを一つのリニアなアドレス空間を持つ共有メモリとすることは、CPU-メモリパス(ネットワーク)のバンド幅を考慮した場合には非現実的であり、従って各プロセッサにローカルメモリを持たせることにより、アクセスパターンコストの抑制を計る必要が出てくる。スレッドを用いた並列処理ではアドレス空間が全てのスレッドにおいて同一の、すなわち共有アドレス空間を前提としており、上記のような理由により、スケーラビリティを得ることが出来ない。すなわちローカルメモリ空間を要求する場合には特別な工夫が必要となり、個々のアプリケーションでこれを実現するのはプログラマの余分な負担になるという欠点がある。また関数の実行を行うサーバとして、通常のプロセス／スレッド生成を用いた場合、かなりのコストが必要とされる。これらのことを考えCLIPではこのサーバを効率良く管理運用するための工夫も行っている。

2.1　マルチプロセッサCLIPerと並列処理

CLIPerは図１に示すように、ローカルメモリとCPUをそれぞれ持つn個のPE(Purocessing Erement)と共有メモリがバスあるいはネットワークによって結合された、マルチプロセッサシステムである。

I/Oを持つPEをマスタ、それ以外のプロセッサをスレーブと呼び、プログラムは全てのPEに全く同じものが開始時にロードされるが、マスタにおいてのみmain()が開始される。

2．2　future処理機構

関数名とその引数をfuture自身の引数として持ち、future関数が呼び出された場合、その返値としてとりあえずfutureが返され、そのあと次のステートメントに実行制御が推移するが、これと平行して、futureサーバプロセスに対して、futureの引数としてその名前が与えれた関数に引数が渡されて評価が開始される。future呼び出し文はfutureの引数となった関数の返値を待たず、ブロックされることはない。futureの値は関数の評価が完了した時点において、関数の返値に置き換わる。futureは関数の評価が完了していなくても、真にその値が必要でない限りfutureの代入を自由に行うことが出来るので、マルチプロセッサにおいてこのfutureのメカニズムを実現することにより、より柔軟な関数単位の並列処理の記述が可能となる。

◆　CLIPerのfuture処理機構(Future Processing Mechanism、以下FPM)

あるPEにおいてfutureが発行されると、FPMはPEを探し関数の処理を依頼するが、もしどのPEも高負荷状態であれば、futureを発行したPEが自分でこの関数を実行するようにする。すなわち、この水平垂直型依頼によりジョブ発行の爆発を防いでいる。

◆WSのfuture

CLIP言語で記述された並列プログラムでは、プログラムが開始時されると、
(1)指定された数の同一のプロセスが生成され、各プロセスはFPMからの関数実行依頼を待つ。
(2)future関数が呼び出されると、その引数として与えられた関数の評価はFPMに対して依頼され、
(3)FPMでは単一キューを通じてfutureの依頼を受理し、選ばれたプロセスに関数の実行を依頼する。
(4)FPMから評価を依頼されたプロセスは、関数の評価が終了すればfutureを発行したところに直接結果を返却し、
(5)FPMに評価の完了を知らせて、再びFPMからの関数実行依頼を待つ。
(6)FPMでは評価の完了を知らせて来たプロセスの負荷状態の更新を行い、次のfutureの依頼に備える。
(7)touch() 関数で、関数の返値の確認が行われれば、関数が未評価であればこのtouch()関数はブロックされ、また評価が完了していればfuture関数の返値は評価された関数の値で与えられる。

3　futureの利用

```
future int I, int add();
main(){
    I=future(add, 1, 2);              ----------(1)
      .       .
     touch(I);                        ----------(2)
     printf("SUM=%d¥n", (int)I);      ----------(3)
}
```

(a)futureのシンタックスその１

```
int   f* I, int  f* add();
main(){
    I= add(1, 2);
}
```

(b)futureのシンタックスその２

図2　futureのシンタックス

我々はfutureのシンタックスとして図3の(a)のように　future の引数として、評価したい関数名とその引数を与えるシンタックスと、(b)のようにfutureを変数型の属性としてとらえたシンタックスの両方について検討した。その結果、(b)の形がC言語のシンタックスとしては自然であり望ましいが、今回については実装の容易さの観点から(a)の形でfutureを実装した。

(a)のステートメント(1)が実行されると、futureの引数としてその名前が与えられた関数 add() が並列処理タスクとして起動され、add(1,2) の評価が開始される。それと同時に、関数 future からの返値としてfutureがただちに返され、(1)の次のステートメントに制御が移る。

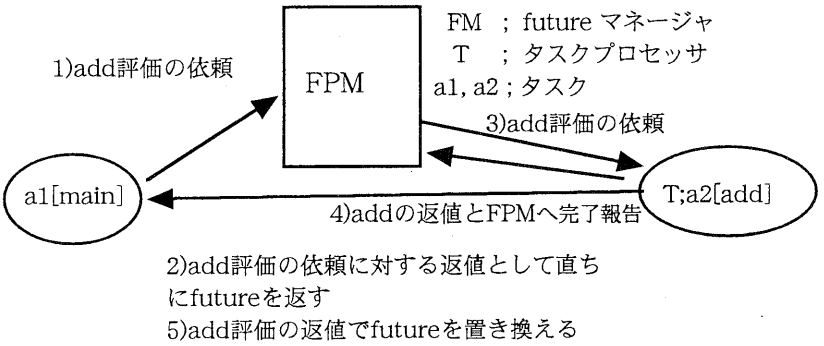

図3　futureの処理過程

このとき平行に処理が行われる。(2)のステートメントに制御が移った場合、add()の評価が完了していなければIにはadd()の返値 future が与えられているが、ここで add() の評価の完了を待つために、touch()関数が呼ばれている。これは引数に与えられた名前を持つ関数の評価の完了を待つための関数である。したがってステートメント(3)に制御が移った時点では、変数 I の値であった future は add(1, 2) の評価後の返値で置き換えられている。

futureの呼び出しは、投機的実行とは異なり返値が必ず用いられることが特徴である。また関数の評価が未完であっても、その返値が実際に必要となるまでは、とりあえずの返値である future をつかって、処理を進めることができる。このことにより、陽に並列実行のコンストラクトを使用する際に必要な、想像力による関数評価と実行制御の時間的シミュレーションを行わずに、アルゴリズムのコーディングを行うことができる。これは並列処理の処理速度の増加以外に、プログラミングパラダイムとしての注目すべきCLIPの特徴である。

4　評価

幾つかのプログラムを使い評価実験を行った環境について述べる。使用した並列処理システムを表1に示す。また、評価に使用したプログラムを表2に示す。

表1　評価に使用した並列処理システム

(1)CLIPer[MC68008x18CPU]
(2)Alpha Server 8200 5/300[DECchip 21164(300MHz)x6CPU]
(3)SUN server1000[SuperSPARC(50MHz)x4CPU]

表2　評価に使用したプログラム

(1)FFT
(2)Nqueen
(3)ペントミノ
(4)巡回セールスマン(12都市)

これらの表2に示すプログラムをCLIP言語で記述し、表1のマルチプロセッサシステムにおいて実

行した結果を以下の図に示す。

○実験結果について

実際の計算時間は、使用したCPUの個数やプロセス個数によって様々な効率で実行される事がわかる。しかし、ここでのこれらの結果について心にとどめておきたいのは、CPUの個数が１個、あるいはプロセスの個数が１個の場合より複数のCPUあるいはプロセスを用いたほうが、遥かに高速に行われることである。また参考までに、それぞれのWSの自動並列オプションをほどこしたコンパイルの結果についても時間測定を行ったが、ほとんどその並列化効果がみられなかった。

ここで重要なのは、CLIP言語を用いた場合容易に並列化されたプログラムを記述できることである。また各図でも示すように、その並列化の効果を得ることが出来ることである。

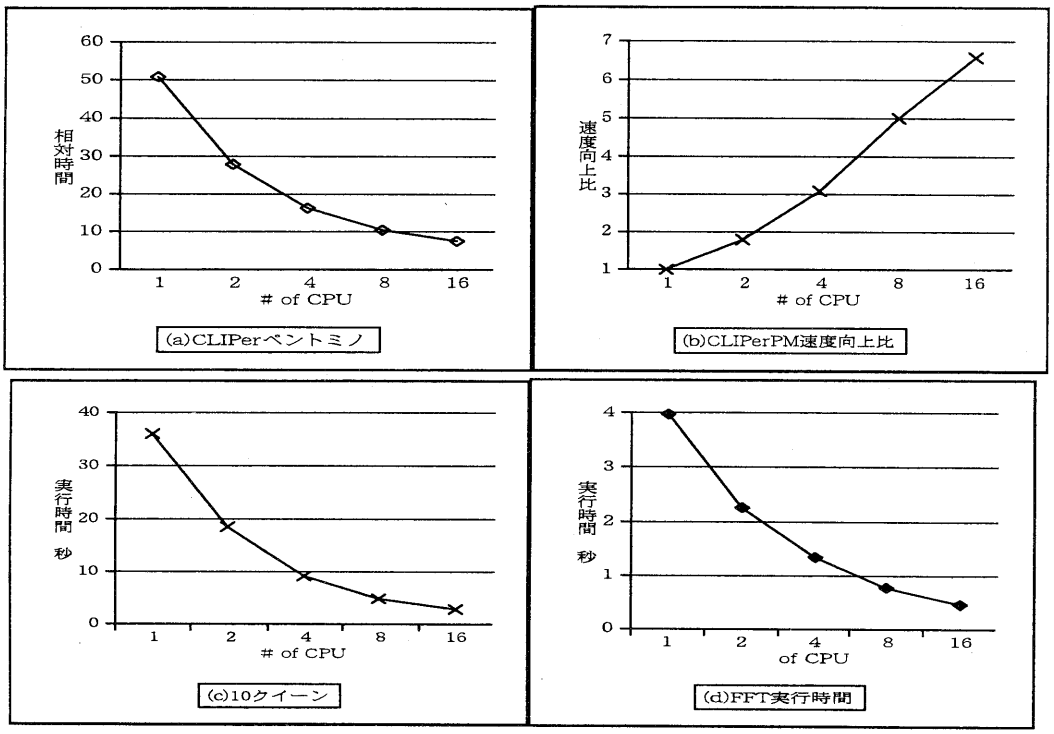

図５-２　CLIPerにおけるテストプログラム

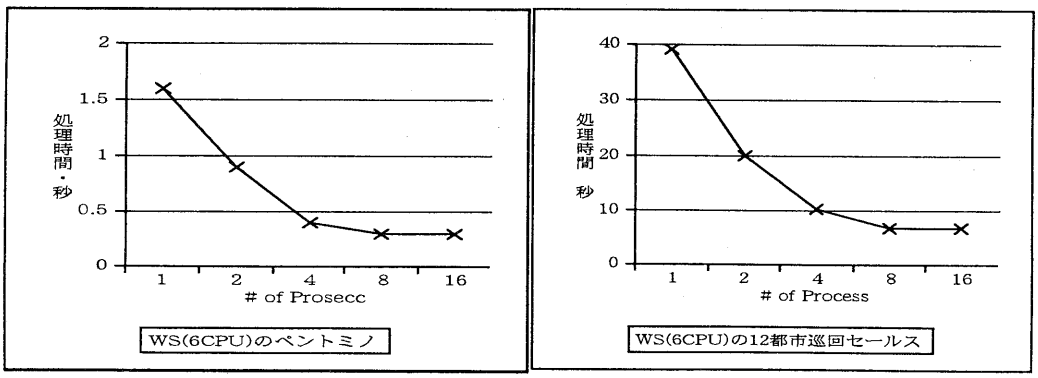

図６　DECアルファサーバ8200における結果

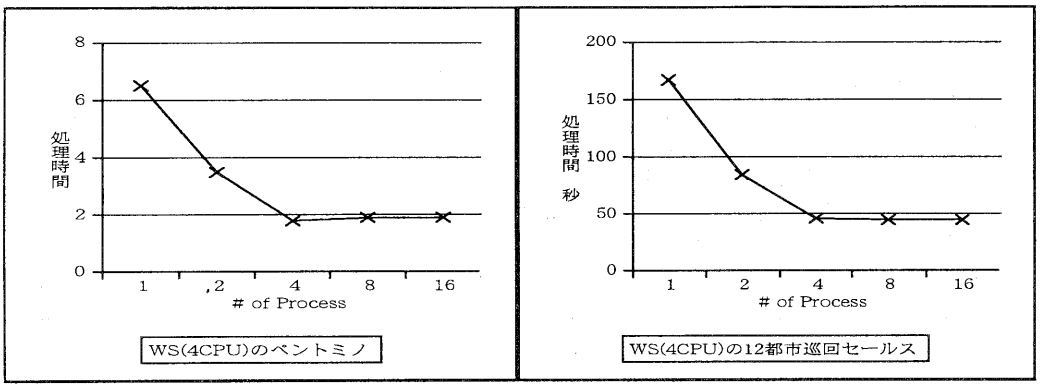

図7　SUN Server1000　における結果

5　まとめ

並列処理言語CLIPをCLIPerおよびワークステーションに実装した。また実際に幾つかのプログラムを使って並列処理の記述が容易に行なえることを示した。また、それらのCLIPで記述されたプログラムを使って処理速度の測定を行い、並列処理が効率的に行われることを示した。

今後は、CLIPのより効率的な実装と、分散処理環境、例えば計算機クラスタへのCLIPの実装をめざす。

参考文献

[1]Robert H, Halstead, Jr."Parallel Symbolic Computing"Computer, Vol. 19, No. 8,pp.35-43(Aug. 1986)
[2]柳瀬龍郎”C言語とfuture関数について”情報処理学会研究報告９５ーSYMー７８(記号処理研究報告N0.78),Vol.95, No. 35, 9-16(1995-3)
[3]中山泰一、松永礼夫、出口光一郎、森下巖、”共有メモリ型並列機における細粒度並列処理のための新しいアクティビティ方式並列実行機構”、情報処理学会論文誌、Vol.34 No.5、985-993(May 1993)
[4]中山泰一、赤沢忠文、野下浩平”ゲーム木の並列探査のための分散的実行管理機構”電子情報通信学会論文誌,Vol.J79-D-I No.9、572-575(1996/9)
[5]山本淳二、鬼頭宏幸、美辺央希、山口喜弘、天野英晴”ローカルメモリをもつマルチプロセッサのソフトウエア環境EULASHのプリコンパイラの評価”、電子情報通信学会論分誌D-1、Vol.J81-D-I No.10、pp1130-1140(1998/10)
[6]Yamamoto, F., Umetani,Y, and Demoto, M.:PARGRAM: A Highi-Level Programming Language for Parallel Processors, Systems and Computers in Japan, Vol. 20, No. 8, pp.100-109(1988)
[7]天津克秀、平田博章、新實治男、柴山潔”分散メモリ型並列計算機向き階層化スレッドスケジューリング方式”、電子情報通信学会論分誌D-I、Vol. J80-D-I No.7、pp615-623(1997/7)
[8]朝倉宏一、渡邊豊英”ワークステーション・クラスタ環境における並列化コンパイラの構成”電気学会論分誌C、118巻4号、pp558-568（平成10年）
[9]Henri E. Bal, M. Frans Kaashoek, Andrew S. Tanenbaum"orca:A Language For Parallel Programming of Distributed Systems", IEEE Trans. Software Eng, Vol.18, No.3, pp190-205(Mar. 1992)
[10]"Splity-C"http://www.cs.ucsb.edu/research/split-c/programs.html